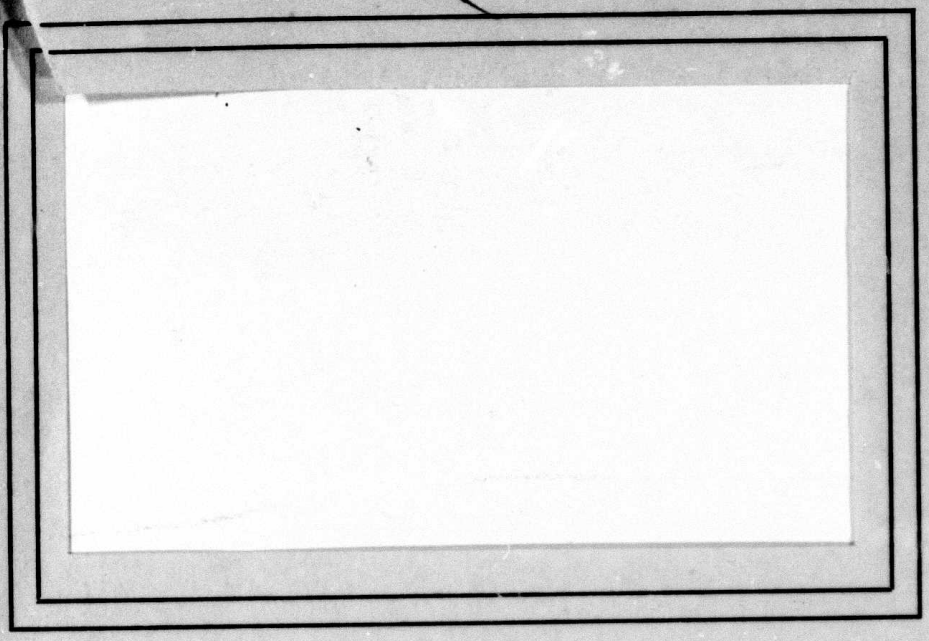
LEVEL II



UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND
20742

(15) DAAG53-76-C-0138,
DARPA Order-3206

(12) (14)

(14) TR-767
DAAG-53-76C-0138

(11) May 1979

(6) REGION REPRESENTATION: QUADTREES FROM BINARY ARRAYS.

(10) Hanan Samet
Computer Science Department
University of Maryland
College Park, MD 20742

(9) Technical rept.

ABSTRACT

An algorithm is presented for constructing a quadtree from the array representation of a binary image. The algorithm examines each pixel in the image once and only once. In addition, as the tree is constructed, only maximal sized nodes are ever created. Thus the algorithm never requires temporary nodes. The execution time of the algorithm is equal to the number of pixels in the image. The amount of space, in addition to that necessary for the final quadtree, is proportional to the log of the image diameter.

The support of the Defense Advanced Research Projects Agency and the U.S. Army Night Vision Laboratory under Contract DAAG-53-76C-0138 (DARPA Order 3206) is gratefully acknowledged, as is the help of Kathryn Riley in preparing this paper. The author has also benefited greatly from discussions with Charles R. Dyer and Azriel Rosenfeld. He also thanks Pat Young for her help with the figures.

1. Introduction

Region representation is important in applications such as image processing, cartography, and computer graphics. Numerous representations are currently being used (see [DRS] for an overview). In this paper we focus our attention on the binary array and quadtree [Klinger] representations. In particular, we present an algorithm for constructing a quadtree from a binary image in a manner that minimizes space requirements during the quadtree construction process. In general, algorithms for transforming one representation into another [DRS, Samet1, Samet2] are important because each representation is well suited for a specific set of operations on an image. The quadtree is useful because it provides a hierarchical representation as well as facilitating operations such as search.

In the remainder of the paper we briefly review the definitions of the representations. This is followed by a description of the algorithm along with motivating considerations. We conclude with some comments about the efficiency of the algorithm. The actual algorithm is given using a variant of ALGOL 60 [Naur].

We assume that the given image is a 2^n by 2^n array of unit square "pixels," each of which has value 0 or 1. The quadtree is an approach to image representation based on successive

subdivision of the array into quadrants. In essence, we repeatedly subdivide the array into quadrants, subquadrants,... until we obtain blocks (possibly single pixels) which consist entirely of either 1's or 0's. This process is represented by a tree of out-degree 4 in which the root node represents the entire array, the four sons of the root node represent the quadrants, and the terminal nodes correspond to those blocks of the array for which no further subdivision is necessary. For example, Figure 1b is a block decomposition of the region in Figure 1a while Figure 1c is the corresponding quadtree. In general, BLACK and WHITE square nodes represent blocks consisting entirely of 1's and 0's respectively. Circular nodes, also termed GRAY nodes, denote non-terminal nodes.

Each node in a quadtree is stored as a record containing six fields. The first five fields contain pointers to the node's father and its four sons, labeled NW, NE, SE, and SW. Given a node P and a son I, these fields are referenced as FATHER(P) and SON(P,I) respectively. The sixth field, named NODETYPE, describes the contents of the block of the image which the represents--i.e., WHITE, BLACK, or GRAY.

2. Algorithm

The quadtree construction algorithm examines each pixel in the binary array once and only once and in a manner which is analogous to a postorder tree traversal. For example, the pixels in the binary array of Fig. 1a are labeled in the order in which they have been examined (e.g., denoting the array by A, we that A[1,1] is examined first, followed by A[1,2], A[2,1], A[2,2], A[1,3],...). However, a node is only created if it is maximal--in other words, if it cannot participate in any further merges (a merge is said to occur when four sons of a node are either all BLACK or all WHITE). For example, Fig. 2a shows the partial quadtree resulting from examining pixels 1, 2, 3, and 4 of Fig. 1a. Note that since all the pixels are not of the same type (i.e., pixels 1, 2, and 3 are WHITE while pixel 4 is BLACK), their nodes cannot participate in any further merges[6] and thus the segment of the final quadtree corresponding to their contribution can be constructed. In contrast, pixels 5, 6, 7, and 8 of Fig. 1a are of the same type (i.e., BLACK) and thus they will be represented by node A in the final quadtree. No nodes are ever constructed corresponding to these pixels. As a final example of a merge, we observe that pixels 17-32 are ultimately represented by node D in the quadtree of Fig. 1c. However, the node corresponding to these pixels is only created once its remaining brothers have been processed (i.e., pixels 33-48 and 49-64). This is in contrast

with the GRAY node corresponding to pixels 1-16 which was created as soon as it was determined that its four sons are not all WHITE or all BLACK.

The main procedure is termed QUADTREE and is invoked with the values of the log of the image diameter (n, for a 2^n by 2^n image array) and the name of the image array. It controls the construction of the quadtree and if the image is all WHITE or all BLACK, then it creates the appropriate one-node tree. The actual construction of the tree is performed by procedure CONSTRUCT which recursively examines all the pixels and creates nodes whenever all four sons are not of the same type. The tree is built as CONSTRUCT returns from examining its sons. CONSTRUCT makes use of a data structure termed pair, denoted by the symbol <,>, which is a record having two fields termed TYPE and POINTER. It is used to return more than one value from CONSTRUCT. COLOR is a function which indicates whether a pixel is BLACK or WHITE.

As an example of the application of the algorithm consider the image given in Fig. 1a. Fig. 1b is the corresponding maximal block decomposition while Fig. 1c is its quadtree representation. The pixels in Fig. 1a have been numbered according to the order in which they are examined by the algorithm. The blocks in Fig. 1b with alphabetic labels correspond to instances where merging has taken place. The alphabetic labels have been

assigned according to the order in which the merged nodes were created (i.e., A, B, C,...). Figs. 2a, 2b, and 2c show the partial quadtrees after pixels 1-4, 1-15, and 49-64 respectively have been examined.

```
node procedure QUADTREE (LEVEL, A);
/* find the quadtree corresponding to a 2↑LEVEL by 2↑LEVEL
   binary array A */
begin
    integer LEVEL;
    global Boolean array A[1:2↑LEVEL, 1:2↑LEVEL]; /*A is a global */
    quadrant I;
    pair P;
    node Q;
    P←CONSTRUCT (LEVEL, 2↑LEVEL, 2↑LEVEL);
    if TYPE(P)=GRAY then
        begin
            FATHER(POINTER(P))←NULL
            return(POINTER(P));
        end
    else
        begin   /* the entire image is BLACK or WHITE */
            Q←CREATENODE();
            NODETYPE(Q)←TYPE(P);
            for I in {NW,NE,SW,SE} do SON(Q,I)←NULL;
            FATHER(Q)←NULL;
            return (Q);
        end;
end;
```

```
pair procedure CONSTRUCT(LEVEL,X,Y);
/* construct the portion of a quadtree of size 2↑LEVEL by
   2↑LEVEL having its southeasternmost pixel corresponding to
   entry A[X,Y] of the image array */
begin
    integer LEVEL,X,Y;
    pair array P[NW...SE];  /* P has entries corresponding
        to NW,NE,SW, and SE */
    quadrant I,J;
    node Q,R;
    if LEVEL=ø then   /* process the pixel */
        return (<COLOR(A[X,Y]),NULL>) /*<,> creates a (POINTER,TYPE) pair */
    else
        begin
            LEVEL←LEVEL-1;
            P[NW]←CONSTRUCT(LEVEL, X-2↑LEVEL, Y-2↑LEVEL);
            P[NE]←CONSTRUCT(LEVEL, X,Y-2↑LEVEL);
            P[SW]←CONSTRUCT(LEVEL, X-2↑LEVEL,Y);
            P[SE]←CONSTRUCT(LEVEL, X,Y);
            if TYPE(P[NW])≠GRAY and
               TYPE(P[NW])=TYPE(P[NE])=TYPE(P[SW])=TYPE(P[SE]) then
                return(P[NW])   /* all brothers are of the
                    same type */
            else
```

```
            begin  /* create a non-terminal GRAY node */
                Q←CREATENODE();
                for I in {NW,NE,SW,SE} do
                    begin
                        if TYPE(P[I])=GRAY then
                        /* link P[I] to its father node */
                            begin
                                SON(Q,I)←POINTER(P[I]);
                                FATHER(POINTER(P[I]))←Q;
                            end
                        else  /* create a maximal node for P[I] */
                            begin
                                R←CREATENODE();
                                NODETYPE(R)←TYPE(P[I]);
                                for J in {NW,NE,SW,SE} do
                                    SON(R,J)←NULL;
                                SON(Q,I)←R
                                FATHER(R)←Q;
                            end;
                    end;
                NODETYPE(Q)←GRAY;
                return(<GRAY,Q>);
            end;
        end;
    end;
```

3. Concluding Remarks

The running time of the quadtree construction algorithm is equal to 4/3 the number of pixels in the image since this is the number of times procedure CONSTRUCT is invoked (and equal to the number of nodes in a complete quadtree for a 2^n by 2^n image). The algorithm is highly recursive. However, the maximum depth of recursion is equal to the log of the image diameter (i.e., n for a 2^n by 2^n image). The algorithm is especially attractive because only nodes with are part of the final quadtree are created. This is in contrast with an approach that would build a complete quadtree for the image and then attempt to obtain maximal blocks by merging. An intermediate approach was used in [Samet2] where a quadtree was constructed for an image given its row-by-row (i.e., raster) description. In that method, the number of nodes was reduced by merging as soon as it became feasible. For example, no merging is possible when processing the first row. However, a merge can be attempted as soon as the first two pixels in the second row are processed. Notice that the method used here is optimal in the sense that a minimum number of nodes is created. This is important when storage is at a premium--i.e., tree nodes require considerably more space than pixels.

4. References

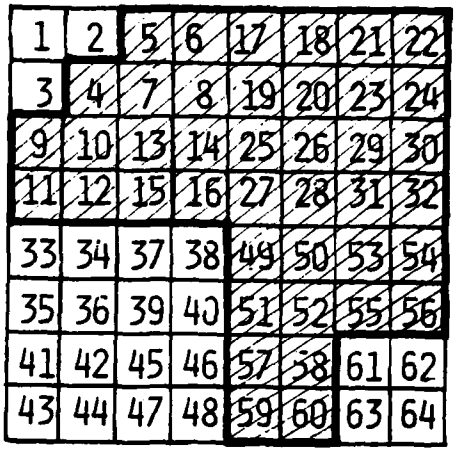
[DRS] C. R. Dyer, A. Rosenfeld, and H. Samet, Region representation: boundary codes from quadtrees, Computer Science TR-732, University of Maryland, College Park, Maryland, February 1979.

[Klinger] A. Klinger and C. R. Dyer, Experiments in picture representation using regular decomposition, Computer Graphics and Image Processing 5, 1976, 68-105.

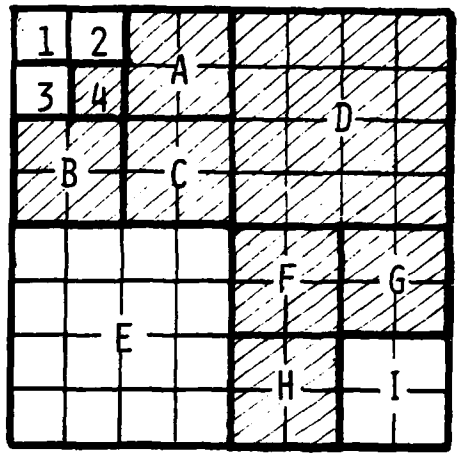
[Naur] P. Naur (Ed.), Revised report on the algorithmic language ALGOL 60, Communications of the ACM 3, 1960, 299-314.

[Samet1] H. Samet, Region representation: quadtrees from boundary codes, Computer Science TR-741, University of Maryland, College Park, Maryland, March 1979.

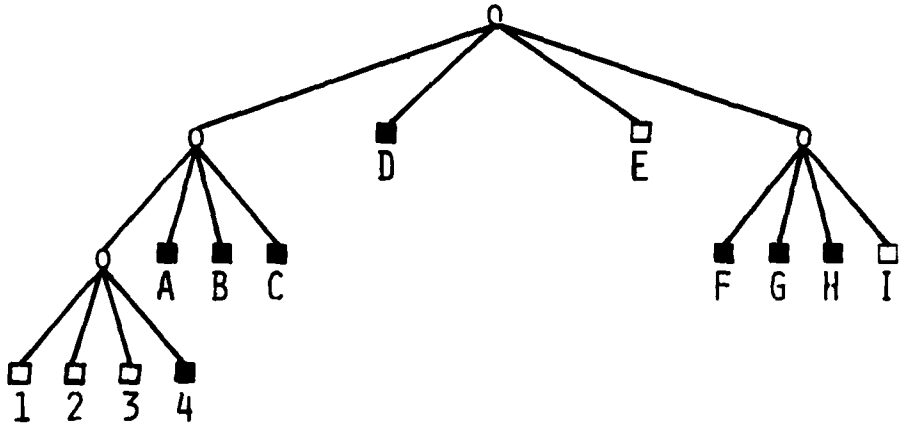
[Samet2] H. Samet, Region representation: raster-to-quadtree conversion, Computer Science TR-766, University of Maryland, College Park, Maryland, May 1979.



a. Sample image



b. Block decomposition of the image in (a)



c. Quadtree representation of the blocks in (b).

Figure 1. An image, its maximal blocks, and the corresponding quadtree. Blocks in the image are shaded.

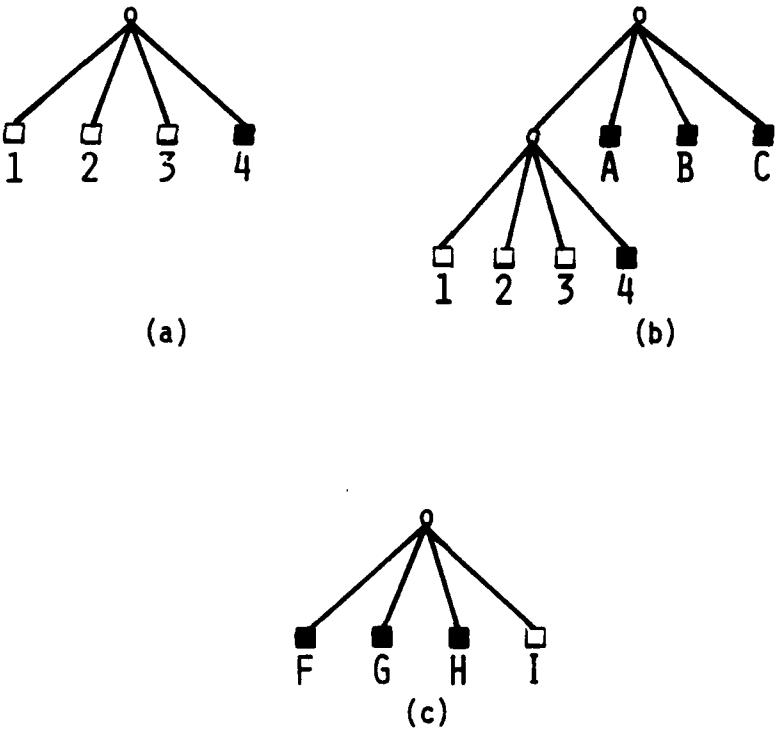


Figure 2. Intermediate trees in the process of obtaining a quadtree corresponding to Figure 1a.